CHAPTER 6

Boundary Conditions

The equations of motion require boundary conditions on all sides of the domain in which the solution is to be obtained, as well as on all surfaces of any objects which lie within the domain. For CFD methods, this implies that boundary conditions must be applied at each face of each computational block. Here, the term boundary conditions is used somewhat loosely as it refers to both physical conditions (e.g. solid wall, symmetry plane, etc.) and interfaces between adjacent or overlapped blocks.

In CFL3D, the boundary condition data are located in the arrays `qi0`(**jdim**,**kdim**,5,4), `qj0`(**kdim**,**idim**-1,5,4), and `qk0`(**jdim**,**idim**-1,5,4) for the i, j, and k directions, respectively. The third index in the arrays indicates the position for the five flow-field primitive variables (Equation (E-3)). The fourth dimension indicates the storage location for the boundary value positions; indices 1 and 2 are the positions for the boundary data at the left face and indices 3 and 4 are the positions for the boundary data at the right face. "Left face" here refers to the i = 1, j = 1, and k = 1 faces. "Right face" refers to the i = **idim**, j = **jdim**, and k = **kdim** faces.

Note that the locations defining the boundary condition regions in "LT14 - I0 Boundary Condition Specification" through "LT19 - KDIM Boundary Condition Specification" are *grid points*, but the actual boundary conditions are applied at *cell-face centers*.

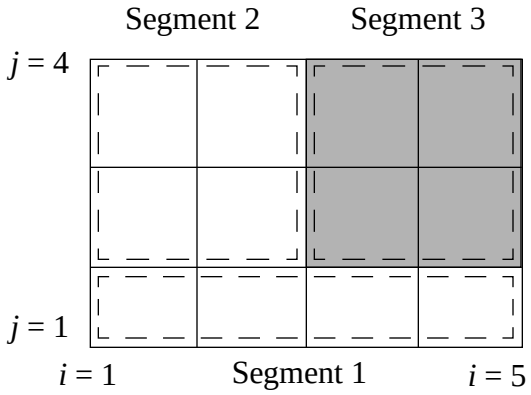

Figure 6-1. Boundary condition segments.

For example, say that Figure 6-1 represents the surface of a wing. At the unshaded cells, it is desired to apply a heated wall boundary condition, while at the shaded cells it is

desired to apply an adiabatic wall boundary condition. One way to accomplish this objective would be to divide the K0 boundary condition into three segments, as indicated by the dashed lines in the figure. The CFL3D input file would look like this:

```
Line Type
 18    K0: GRID   SEGMENT    BCTYPE      ISTA      IEND      JSTA      JEND      NDATA
             1          1      2004         1         5         1         2          2
         twtype       Cq
           1.6      0.0
             1          2      2004         1         3         2         4          2
         twtype       Cq
           1.6      0.0
             1          3      2004         3         5         2         4          2
         twtype       Cq
           0.0      0.0
```

Note that, for example, the *grid points* $i = 1$ to 5, $j = 1$ to 2 define the boundary of the *cells* at which the boundary condition type for segment 1 is applied.

Two types of boundary condition representations are employed in CFL3D, namely, cell-center and cell-face. For cell-center type boundary conditions, the flow-field variables are specified at "ghost" points corresponding to two cell-center locations analytically extended outside the grid as illustrated in Figure 6-2. For example, qi0(j,k,**l**,1) contains the boundary data at the ghost point locations right next to the $i = 1$ cell-center data and qi0(j,k,**l**,2) contains the boundary data in the second set of ghost point locations at the left boundary. Likewise, qi0(j,k,**l**,3) contains the boundary data at the ghost point locations right next to the $i =$ **idim** cell-center data and qi0(j,k,**l**,4) contains the boundary data in the second set of ghost point locations at the right boundary. The same boundary data location definitions apply for the qj0 and qk0 arrays.

For cell-face type boundary conditions, the flow-field variables and their gradients are specified at the cell-face boundary as illustrated in Figure 6-3. For cell-face type boundary conditions, qi0(j,k,**l**,1) contains the flow-field variable data at the $i = 1$ grid cell-face and qi0(j,k,**l**,2) contains the flow-field variable gradients at the $i = 1$ grid cell-face. Likewise, qi0(j,k,**l**,3) contains the flow-field variable data at the $i =$ **idim** grid cell-face and qi0(j,k,**l**,4) contains the flow-field variable gradients at the $i =$ **idim** grid cell-face. The same boundary data location definitions apply for the qj0 and qk0 arrays.

Note that when running in 2-d mode (**i2d** = 1), it *does not matter* what boundary conditions are used on the $i = 1$ and $i =$ **idim** faces (provided that the boundary condition itself does not require more than one interior cell). **Bctype** is generally set to 1002 on these faces.

Boundary conditions for the eddy viscosity are stored in vi0, vj0, and vk0, while boundary conditions for the field equation turbulence models are stored in ti0, tj0, and tk0. Both of these are cell-center type, although only the first ghost points (1 and 3) are used. The field equation turbulence model boundary conditions are described in more detail in Section H.7 on page 296.

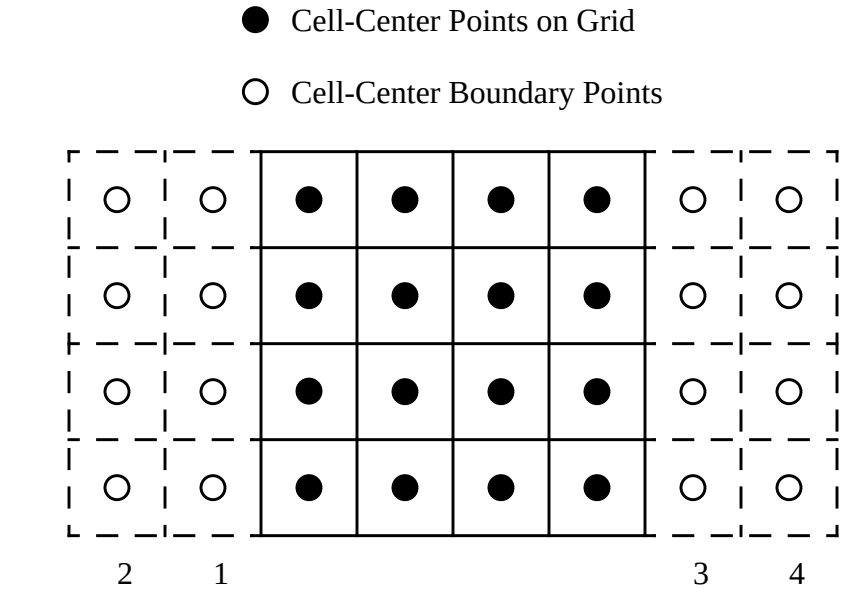

Figure 6-2. Cell-center type boundary locations.

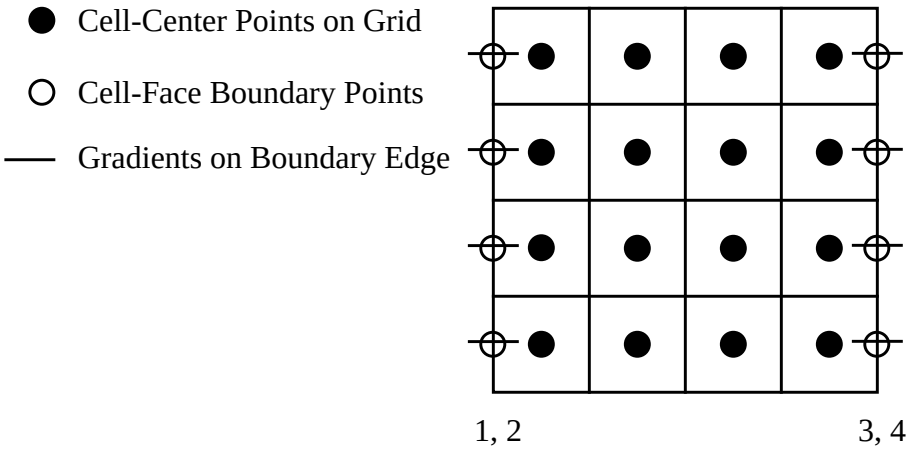

Figure 6-3. Cell-face type boundary conditions.

6.1 Physical Boundary Conditions

The 1000 series of boundary condition types are used for physical boundary conditions and are specified at any of a block's six faces in "LT14 - I0 Boundary Condition Specification" on page 32 through "LT19 - KDIM Boundary Condition Specification" on page 35. In each case, **ndata** = 0, since no additional information is required for implementation of the condition. The 1000 series boundary condition types currently supported are as follows:

bctype	boundary condition
1000	free stream
1001	general symmetry plane
1002	extrapolation
1003	inflow/outflow
1004	(no longer available, use 2004 instead)
1005	inviscid surface
1008	tunnel inflow
1011	singular axis – half-plane symmetry
1012	singular axis – full plane
1013	singular axis – partial plane

6.1.1 Free Stream

bctype 1000

The free stream boundary conditions are cell-center type boundary conditions. The five flow-field variables for both sets of ghost points are set equal to the initial values, which are:

$$\begin{aligned} \rho_{initial} &= 1.0 \\ u_{initial} &= M_\infty \cos\alpha\cos\beta \\ v_{initial} &= -M_\infty \sin\beta \\ w_{initial} &= M_\infty \sin\alpha\cos\beta \\ p_{initial} &= \rho_{initial}(a_{initial})^2/\gamma \end{aligned} \qquad \textbf{(6-1)}$$

where $a_{initial} = 1.0$.

6.1.2 General Symmetry Plane

bctype 1001

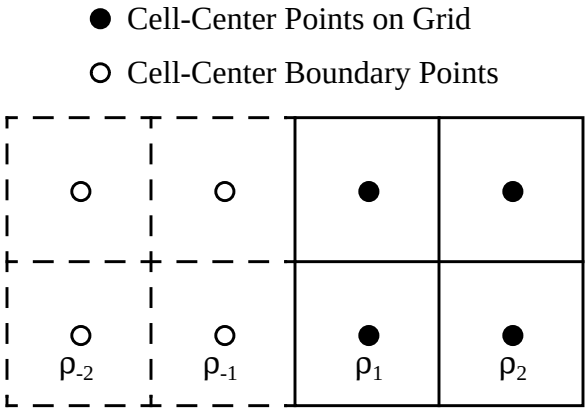

Figure 6-4. Symmetry boundary conditions (**bctype** 1001) for density.

The symmetry plane boundary conditions are cell-center type boundary conditions. As the name implies, symmetry is assumed across an axis. The ghost point density values are set equal to their "mirror image" counterparts. With the points defined as in Figure 6-4,

$$\begin{aligned} \rho_{-1} &= \rho_1 \\ \rho_{-2} &= \rho_2 \end{aligned} \tag{6-2}$$

(Note that ρ_{-1} = `qi0`(j,k,1,1) at the $i = 1$ face, for example.) The pressure values are assigned in the same way.

The velocity components at the ghost cells are obtained as follows. Consider ghost cells at an $i = 1$ face. Note that the normalized contravariant velocity $\overline{U}$ is normal to an i = constant face. Let $\overline{U}_1$ be the normalized contravariant velocity at cell center 1 in Figure 6-4. For an i symmetry plane, $\overline{U}$ must have opposite signs on each side of the plane. Thus

$$\begin{aligned} u_{-1} &= u_1 - 2\hat{\xi}_x \overline{U}_1 \\ v_{-1} &= v_1 - 2\hat{\xi}_y \overline{U}_1 \\ w_{-1} &= w_1 - 2\hat{\xi}_z \overline{U}_1 \end{aligned} \tag{6-3}$$

where $\hat{\xi}_x$, $\hat{\xi}_y$, and $\hat{\xi}_z$ are the metrics (unit normals) at the $i = 1$ face.

Since $(\hat{\xi}_x)^2 + (\hat{\xi}_y)^2 + (\hat{\xi}_z)^2 = 1$, $\bar{U}_{-1} = \bar{U}_1$. The velocity components at ghost cell center –2 are set in a similar manner using the data at cell center 2. (Note that this is now a "general" symmetry condition, applicable to any plane, not just the $x-z$ plane as in earlier versions of the code. Consequently, there is no longer a **bctype** 1006.)

6.1.3 Extrapolation

bctype 1002

The extrapolation boundary conditions are cell-center type boundary conditions. The ghost points are extrapolated from the computational domain. Based on the locations of ρ_1, ρ_{-1} and ρ_{-2} depicted in Figure 6-4, the extrapolated values would be

$$\begin{aligned} \rho_{-1} &= \rho_1 \\ \rho_{-2} &= \rho_1 \end{aligned} \tag{6-4}$$

The same zeroth order extrapolation is used for the boundary values of the other four flowfield variables.

6.1.4 Inflow/Outflow

bctype 1003

The inflow/outflow boundary conditions are cell-face type boundary conditions. In the far field, the velocity normal to the far boundary (pointing *out* of the grid) and the speed of sound are obtained from two locally 1-d Riemann invariants:

$$R \equiv \bar{u} \pm \frac{2a}{\gamma - 1} \tag{6-5}$$

where the boundary is considered a surface of constant ξ (in this example) with decreasing ξ corresponding to the interior of the domain and

$$\bar{u} = \bar{U} - \frac{\xi_t}{|\nabla\xi|} \tag{6-6}$$

$$\bar{U} = \frac{\xi_x}{|\nabla\xi|}u + \frac{\xi_y}{|\nabla\xi|}v + \frac{\xi_z}{|\nabla\xi|}w + \frac{\xi_t}{|\nabla\xi|} \tag{6-7}$$

R^- can be evaluated locally from conditions outside the computational domain and R^+ can be determined locally from inside the domain. The normal velocity and speed of sound are determined from

$$\bar{u}_{face} = \frac{1}{2}(R^{+} + R^{-})$$
$$a_{face} = \frac{\gamma - 1}{4}(R^{+} - R^{-}) \qquad (6\text{-}8)$$

The Cartesian velocities are determined by decomposing the normal and tangential velocity vectors:

$$u_{face} = u_{ref} + \frac{\xi_x}{|\nabla\xi|}(\bar{u}_{face} - \bar{u}_{ref})$$
$$v_{face} = v_{ref} + \frac{\xi_y}{|\nabla\xi|}(\bar{u}_{face} - \bar{u}_{ref}) \qquad (6\text{-}9)$$
$$w_{face} = w_{ref} + \frac{\xi_z}{|\nabla\xi|}(\bar{u}_{face} - \bar{u}_{ref})$$

For inflow, $ref \Rightarrow \infty$. For outflow, ref represents the values from the cell inside the domain adjacent to the boundary.

The sign of the normal velocity $\bar{U}_{face} = \bar{u}_{face} + \xi_t/|\nabla\xi|$ determines whether the condition is at inflow ($\bar{U}_{face} < 0$) or outflow ($\bar{U}_{face} > 0$). The entropy p/ρ^{γ} is determined using the value from outside the domain for inflow and from inside the domain for outflow. The entropy and speed of sound are used to determine the density and pressure on the boundary:

$$\rho_{face} = \left[\frac{(a_{face})^2}{\gamma s_{face}}\right]^{\frac{1}{\gamma - 1}}$$
$$p_{face} = \frac{\rho_{face}(a_{face})^2}{\gamma} \qquad (6\text{-}10)$$

Note that when **i2d** = –1, the 1003 boundary condition is augmented by the far-field point-vortex correction[38] (for 2-d, $x - z$ plane only).

6.1.5 Inviscid Surface

bctype 1005

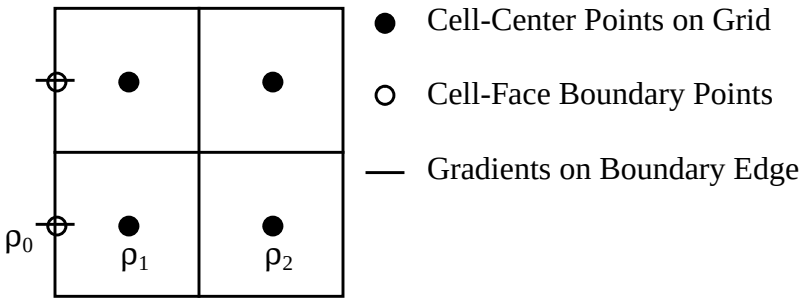

Figure 6-5. Inviscid surface boundary condition (**bctype** 1005) for density.

The inviscid surface boundary conditions are cell-face type boundary conditions. The cell-face boundary values for density are approximated with the density values of the nearest cell-center points on the grid. With the points defined as in as Figure 6-5,

$$\rho_0 = \rho_1 \tag{6-11}$$

The boundary values for pressure are obtained in the same manner.

The velocity component normal to the surface is set to zero in the following manner. Assume that the surface in Figure 6-5 is a k = constant surface, for which the metrics (unit normals) are $\hat{\zeta}_x$, $\hat{\zeta}_y$, and $\hat{\zeta}_z$. Then the normalized contravariant velocity (normal to the k direction) at cell center 1 is

$$\overline{W}_1 = u_1\hat{\zeta}_x + v_1\hat{\zeta}_y + w_1\hat{\zeta}_z + \hat{\zeta}_t \tag{6-12}$$

The velocity components at the wall are then calculated using

$$\begin{aligned} u_0 &= u_1 - \hat{\zeta}_x\overline{W}_1 \\ v_0 &= v_1 - \hat{\zeta}_y\overline{W}_1 \\ w_0 &= w_1 - \hat{\zeta}_z\overline{W}_1 \end{aligned} \tag{6-13}$$

Using $(\hat{\zeta}_x)^2 + (\hat{\zeta}_y)^2 + (\hat{\zeta}_z)^2 = 1$, it may be seen that $\overline{W}_0 = 0$ so that the normal velocity at the wall is zero.

The gradient values for all five flow-field variables needed at the cell-face are obtained using two-point differences. For example, for density,

$$\nabla\rho = 2(\rho_1 - \rho_0) \tag{6-14}$$

6.1.6 Constant Enthalpy and Entropy Inflow

bctype 1008

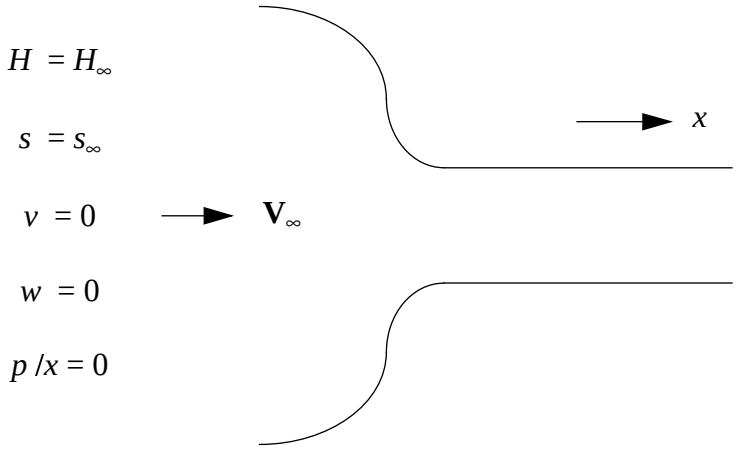

Figure 6-6. Constant enthalpy and entropy inflow boundary conditions (**bctype** 1008).

The constant enthalpy and entropy boundary condition (sometimes referred to as the wind tunnel inflow boundary condition) is a cell-center type condition. A grid set up to use the tunnel inflow condition should always have the x coordinate running along the length of the tunnel as shown in Figure 6-6. It is assumed that the entropy and the enthalpy are at the free-stream conditions. The v and w components of velocity are set to zero and the pressure gradient at the boundary is also zero. The density at the boundary is obtained from

$$s_\infty = \frac{p}{\rho^\gamma} \tag{6-15}$$

The u component of velocity is obtained from

$$H_\infty = \frac{a^2}{\gamma - 1} + \frac{u^2}{2} \tag{6-16}$$

The zero pressure gradient condition is used to set the pressure in the ghost cell equal to the nearest interior value.

For running internal (wind tunnel type) flows, **bctype** 2003 is usually preferred as the inflow boundary condition over 1008, although both may work. For the corresponding wind tunnel outflow boundary condition, **bctype** 2002 should generally be used.

6.1.7 Singular Axis Using Half-plane Symmetry

bctype 1011

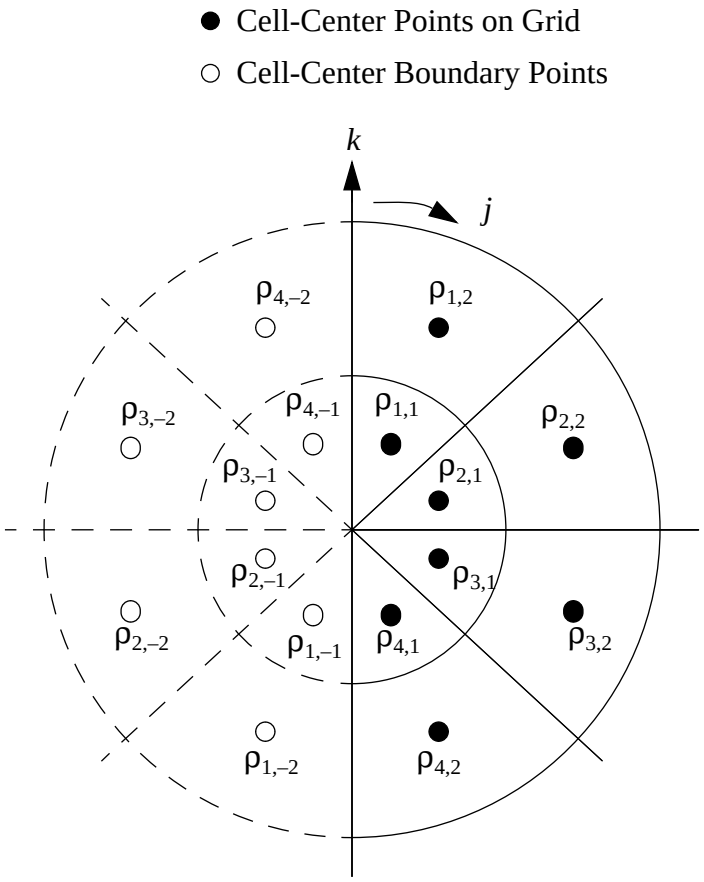

Figure 6-7. Singular axis with half-plane symmetry boundary condition (**bctype** 1011).

The singular axis using half-plane symmetry boundary conditions are cell-center type boundary conditions. As an example of this boundary condition, assume the singular axis occurs at $k = 1$. If **jdim** = 5, an i = constant plane might look like the one drawn in Figure 6-7. In the figure, the subscripts for ρ are j and then k (i.e. $\rho_{j,k}$). The boundary values for density are assigned as

$$\begin{aligned} \rho_{1,-1} &= \rho_{4,1} \qquad \rho_{1,-2} = \rho_{4,2} \\ \rho_{2,-1} &= \rho_{3,1} \qquad \rho_{2,-2} = \rho_{3,2} \\ \rho_{3,-1} &= \rho_{2,1} \qquad \rho_{3,-2} = \rho_{2,2} \\ \rho_{4,-1} &= \rho_{1,1} \qquad \rho_{4,-2} = \rho_{1,2} \end{aligned} \tag{6-17}$$

The other four flow-field variables are assigned in a similar fashion; however, the normalized contravariant velocities and metrics are used to determine the correct signs for the

velocity components in a manner similar to boundary condition type 1001. First note that by the assumption of half-plane symmetry, the direction cosines on $j = 1$ are the same as $j =$ **jdim** apart from the sign; let these metrics be denoted $\hat{\eta}_x$, $\hat{\eta}_y$, and $\hat{\eta}_z$. Then, for example,

$$\overline{V}_{4,1} = u_{4,1}\hat{\eta}_x + v_{4,1}\hat{\eta}_y + w_{4,1}\hat{\eta}_z + \hat{\eta}_t \qquad \textbf{(6-18)}$$

$$\begin{aligned} u_{1,-1} &= u_{4,1} - 2\hat{\eta}_x \overline{V}_{4,1} \\ v_{1,-1} &= v_{4,1} - 2\hat{\eta}_y \overline{V}_{4,1} \\ w_{1,-1} &= w_{4,1} - 2\hat{\eta}_z \overline{V}_{4,1} \end{aligned} \qquad \textbf{(6-19)}$$

6.1.8 Singular Axis Using Full Plane, Flux Specified

bctype 1012

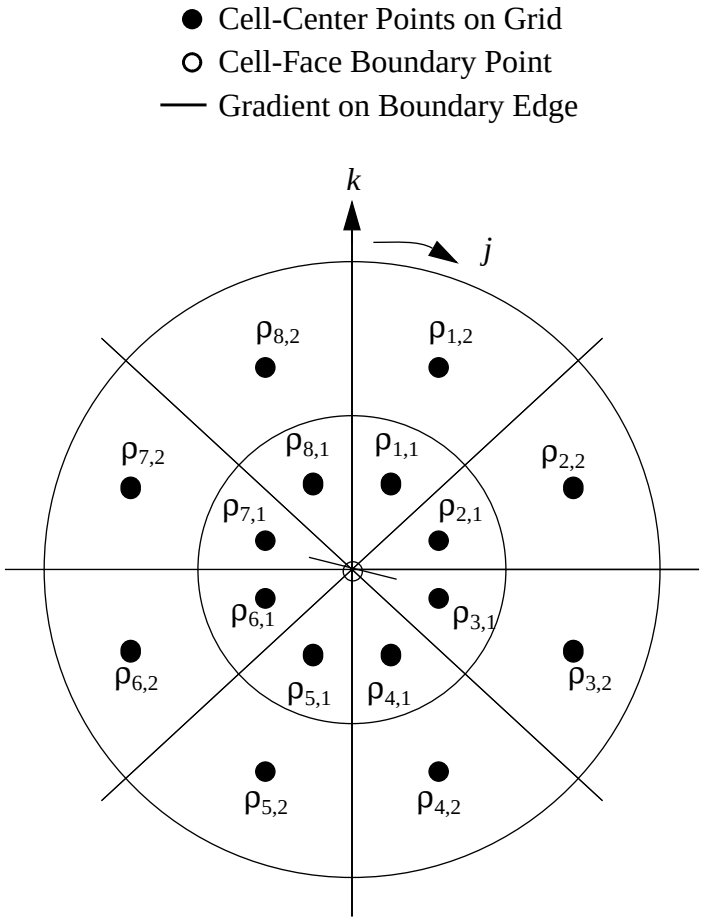

Figure 6-8. Singular axis using full plane boundary condition (**bctype** 1012).

The singular axis using full plane boundary conditions are cell-face type boundary conditions. As an example of this boundary condition, assume the singular axis occurs at $k = 1$. If **jdim** = 9, an i = constant plane might look like the one drawn in Figure 6-8. In the figure, the subscripts for ρ are j and then k (i.e. $\rho_{j,k}$).

For the cell-face boundary point value of density (ρ_0), an average value of all the values at $k = 1$ is obtained, i.e.

$$\rho_{j,0} = \frac{\sum_{j=1}^{\text{jdim}-1} \rho_{j,1}}{\text{jdim}-1} \qquad \textbf{(6-20)}$$

The flux value is obtained with a two point extrapolation using ρ_1 and ρ_0:

$$(\nabla\rho)_j = 2(\rho_{j,1} - \rho_{j,0}) \qquad \textbf{(6-21)}$$

The boundary values for all five flow-field variables are assigned as described for density.

A known problem exists when using this boundary condition with the Baldwin-Lomax turbulence model. In such cases, the code employs the “wall” Baldwin-Lomax equation option rather than the “wake” Baldwin-Lomax equation option at the 1012 boundary.

6.1.9 Singular Axis Using Extrapolation (Partial Plane)

bctype 1013

The singular axis using extrapolation boundary conditions are cell-center type boundary conditions. The ghost points are extrapolated from the computational domain. This boundary condition is used with singular axes for which neither boundary condition type 1011 or 1012 is appropriate (quarter planes, for instance). For example, the density boundary values would be approximated as

$$\begin{aligned} \rho_{-1} &= \rho_1 \\ \rho_{-2} &= \rho_1 \end{aligned} \qquad \textbf{(6-22)}$$

The same first order extrapolation is used for the boundary values of the other flow-field variables.

6.1.10 A Word About Singular Metrics

Version 5.0 will automatically detect collapsed grid lines (e.g. singular metrics, cell faces with zero area) on block boundaries. (Collapsed grid lines in the “interior” of a block

are not allowed.) The detection of singular metrics is no longer keyed to the specification of either **bctype** 1011, 1012, or 1013. While these boundary conditions are still applicable, the new singular metric detection allows any other standard boundary condition that does not rely on the grid metrics at the boundary to be used as well. Note that the following boundary conditions rely on the grid metrics at the boundary and so should not be used on a singular face/face segment: 1001 (symmetry), 1005 (inviscid), 1003 (inflow/outflow), 2003 (inflow with specified total conditions) and 2006 (radial equilibrium).

When CFL3D detects singular metrics on a particular block, a message is written to the main output file (unit 11) indicating the location. It is always a good idea to verify that any singular block faces/face segments are in fact correctly detected. Metrics are treated as singular if the total area of a face/face segment is less than a parameter `atol` (set near the top of subroutine `metric`, in module `lbcx.f`). In certain cases, `atol` may need to be changed (increased in magnitude if faces that are known to be non-singular are flagged as singular and decreased if faces that are known to be singular are not flagged as such, in which case the code will give floating point overflow in subroutine `metric`).

6.2 Physical Boundary Conditions with Auxiliary Data

For the “2000” series of boundary conditions, auxiliary data is required. Therefore, **ndata** 0. The following sections describe the boundary condition types for constant input data (**ndata** > 0). For these conditions, the information in “LT14 - I0 Boundary Condition Specification” on page 32 through “LT19 - KDIM Boundary Condition Specification” on page 35 is immediately followed by a header line, then a single data line with the **ndata** values appropriate for the particular boundary condition. Section 6.2.8 describes how to use the same boundary condition types with variable input data.

Input values for **bctype** (i.e. boundary condition types currently supported) as follows:

bctype	boundary condition
2002	specified pressure ratio
2003	inflow with specified total conditions
2004	no-slip wall
2005	periodic in space
2006	set pressure to satisfy the radial equilibrium equation
2007	set all primitive variables
2102	pressure ratio specified as a sinusoidal function of time

6.2.1 Specified Pressure Ratio

bctype 2002

The specified pressure ratio boundary condition, generally used as the outflow boundary condition for internal flows, is a cell-center type boundary condition. A single pressure ratio, $\tilde{p}/\tilde{p}_\infty$, is specified on input. The parameter **ndata** must be set to 1 for the input pressure ratio value to be read. This pressure ratio is used to set both cell-center pressure boundary values (p_{-1} and p_{-2}). Extrapolation from inside the computational domain is used to set the boundary values for ρ, u, v, and w. See "Extrapolation" on page 84. An example of the input lines is:

```
p/pinf
 0.910
```

When using **bctype** 2002 as the outflow condition for internal (wind tunnel type) flows, the tunnel Mach number and/or mass flow should be monitored and **p/pinf** adjusted accordingly to obtain the correct conditions. This is usually an iterative process.

6.2.2 Inflow With Specified Total Conditions

bctype 2003

The inflow with specified total conditions boundary condition (sometimes referred to as an "engine inflow" boundary condition because it is often used to specify the conditions at an inflow face where an engine exhaust is located) is a cell-center type boundary condition. The following five pieces of information are provided on input (**ndata** = 5): an estimated inflow Mach number (M_e), the total pressure ratio ($\tilde{p}_t/\tilde{p}_\infty$), the total temperature ratio ($\tilde{T}_t/\tilde{T}_\infty$), and the flow directions (α and β) in degrees. These values are used as the external state in a 1-d characteristic boundary condition. An example of the input lines is:

```
  Mach       Pt/Pinf  Tt/Tinf   Alphae    Betae
 0.300         4.000  1.17555      0.0      0.0
```

Boundary condition type 2003 is very similar to boundary condition type 1003 in that either interior or exterior values are chosen, depending on the sign of the characteristics at the boundary. One difference between them is that, while 1003 uses far-field reference-state levels for the exterior values, 2003 uses the total temperature, pressure, Mach number, and flow angle that are input to determine the reference exterior values. Another difference is the way the density and pressure boundary condition values are determined. Boundary condition type 1003 calculates these values as shown in Equation (6-10). Boundary condition type 2003 first determines a Mach number at the boundary using the velocities and speed of sound at the boundary (which were calculated through the characteristic method):

$$M^2 = \frac{u^2 + v^2 + w^2}{a^2} \tag{6-23}$$

It should be noted that the inflow Mach number may end up slightly different from the input M_e for a converged solution. The pressure and density boundary conditions are then determined with

$$p = \frac{\tilde{p}_t / \tilde{p}_\infty}{\gamma\left(1 + \frac{\gamma - 1}{2} M^2\right)^{\gamma/(\gamma-1)}} \tag{6-24}$$

$$\rho = \frac{\gamma p}{a^2}$$

The boundary condition velocity components are determined the same way as for boundary condition type 1003.

6.2.3 Viscous Surface

bctype 2004

The viscous surface boundary conditions are cell-face type boundary conditions. The no-slip condition ($\mathbf{V} = 0$) is applied at the surface. Two pieces of auxiliary information are supplied on input (**ndata** = 2): the wall temperature ($\tilde{T}_w / \tilde{T}_\infty$) and the mass flow ($C_q$), where $C_q = (\rho u_{normal}) / (\rho u)_\infty$. ($C_q$ is zero if there is no flow through the wall.) An example of the input lines is:

```
Twtype          Cq
 0.95        -0.05
```

where

Twtype > 0 sets $\tilde{T}_w / \tilde{T}_\infty$ = **Twtype**
Twtype = 0 sets adiabatic wall
Twtype < 0 sets $\tilde{T}_w$ at the stagnation temperature
Cq < 0 results in suction (mass flow OUT of the zone)

Cq > 0 results in blowing (mass flow INTO the zone)

Cq = 0 results in no flow through the wall (same as old **bctype** 1004)

Note that for a dynamic grid, **bctype** 2004 gives no-slip relative to the moving wall. To set the wall velocity to zero relative to a non-moving reference frame when the mesh is moving, use –2004 instead of 2004; this option makes sense only if the grid motion is tangential to the surface.

Also note that boundary condition type 2004 supersedes boundary condition types 1004 *and* 2004 in previous versions of CFL3D. The main reason for the replacement is that 1004 relied on global parameters **isnd** and **c2spe** to determine whether adiabatic wall or constant-temperature wall conditions were invoked. As a consequence, all 1004 segments had to be the same. Now, with 2004 (along with its additional data field **Twtype**), every no-slip wall segment can be set with different wall temperature conditions, if desired. Boundary condition type 2004 also allows for mass flow through the wall (suction or blowing) through the second additional data field **Cq**.

The no-slip wall boundary condition is implemented as follows:

- The nondimensional pressure on the body p_b is determined through linear extrapolation:

$$p_b = p_1 - (p_2 - p_1)/2 \tag{6-25}$$

  But if $p_b < 0$, then $p_b = p_1$. Here the indices 1 and 2 indicate the first and second cell-center values away from the wall, respectively.
- The nondimensional square of the speed of sound, variable `c2`, at the wall is next determined. (Note that `c2` = $(\tilde{a}_w/\tilde{a}_\infty)^2 = \tilde{T}_w/\tilde{T}_\infty$.)

  If **Twtype** > 0, `c2` = **Twtype**.

  If **Twtype** < 0, `c2` = $1 + \frac{\gamma - 1}{2}(M_\infty)^2$.

  If **Twtype** = 0, `c2` = $\left(\frac{a_1}{a_\infty}\right)^2\left[1 + \frac{\gamma - 1}{2}(M_1)^2\right]$.
- The surface velocities are then determined as

$$\begin{aligned} u_w &= M_\infty C_q \frac{\xi_x}{|\nabla\xi|}\frac{(c2)}{\gamma p_b} + u_{mesh} \\ v_w &= M_\infty C_q \frac{\xi_y}{|\nabla\xi|}\frac{(c2)}{\gamma p_b} + v_{mesh} \\ w_w &= M_\infty C_q \frac{\xi_z}{|\nabla\xi|}\frac{(c2)}{\gamma p_b} + w_{mesh} \end{aligned} \tag{6-26}$$

where u_{mesh}, v_{mesh}, and w_{mesh} are the velocity components of the mesh (they equal 0 if the grid/body is not in motion).

- Since 2004 is a cell-face boundary condition type (i.e. the boundary conditions and their gradient are applied at the cell face rather than in ghost cells),

$$\rho_w = \frac{\gamma p_b}{c2}$$
$$p_w = p_b \qquad \textbf{(6-27)}$$

The gradients at the wall for all the primitive variables are determined via $\nabla\rho = 2\rho_1 - 2\rho_w$, etc.

Note that `smin` is computed from viscous walls *only.* If a wall is inviscid, then as far as `smin` is concerned, it is invisible. This is important to remember when viscous boundary conditions are turned on after running a case inviscidly for some time since `smin` may never have been computed!

6.2.4 Periodic (In Space)

bctype 2005

The periodic boundary conditions are cell-center type boundary conditions. Four pieces of additional information must be input (**ndata** = 4). The number of the grid with which the current grid is periodic and the rotation angle ($d\theta_x, d\theta_y, d\theta_z$) about one of the coordinate axes (x, y, z) are needed. Only *one* of the three angles can be used at a time and the other two angles *must* be identically zero. The angles should be determined from the right-hand rule. For example, if rotation is desired about the N axis (where N is either x, y or z, point the right-hand thumb in the direction of the +N axis. The fingers will curl in the direction of the positive angle. When setting the angle for a particular face (i.e. the $i = 1$ face), set the angle of rotation equal to the angle that the *periodic face* would have to move through to get to this face.

For a sample input, assume the current face on which the boundary condition is being set is $j = 1$. The the following input will cause the current face to be periodic with the **jdim** face in grid 2, where a rotation of +45 degrees about the x axis would map the **jdim** face of grid 2 onto the $j = 1$ face of the current grid:

```
ngridp          dthx      dthy      dthz
     2          45.0       0.0       0.0
```

At present, it is assumed that the current block and the block with which it is periodic match 1-1 at their corresponding faces after the rotation. Note that there is *not* a check for this! Also, the two blocks are assumed to be aligned similarly. That is i, j, and k on the first block must be defined *exactly* the same on the second block. For example, if the

k = **kdim** face of the current block has the periodic boundary condition applied to it, it is implicitly assumed that it is periodic with the $k = 1$ face of the specified second block *and* i and j run in the exact same directions on both blocks. Note that this means that two of the dimensions (**idim** and **jdim** in the example) must be the same on both blocks.

The periodic boundary condition also works for a grid with one cell (two grid planes) in the periodic direction if the grid is set to be periodic with itself. Note, however, that if a particular block is periodic with a *different* block, then neither block should be only one cell wide in the periodic direction.

In addition, the periodic boundary condition may be used for linear displacement, provided the rotation angles are set to zero. Alternatively, the 1-1 block connection can be used (linear displacement only!). The 1-1 internal check will flag geometric mismatches, which should be equal to the desired linear periodic displacement. This is a good check of the input which is not available with boundary condition type 2005.

Boundary condition type 2005 is a limited-use periodic (in space) boundary condition. For example, if one is solving for flow through a duct and it is known that the solution is periodic over 90 degrees (i.e. the solution is identical in each of the four quadrants of the duct), then a solution need only be obtained on a grid spanning 90 degrees, with periodic boundary conditions applied at the two edges.

For an illustration of how this boundary condition works, consider Figure 6-9. This example is for flow through a 90 degree wedge (the flow direction is in the third dimension, out of the page). The flow is periodic over 90 degrees, so periodic boundary conditions are applied at the $j = 0$ and $j =$ **jdim** faces.

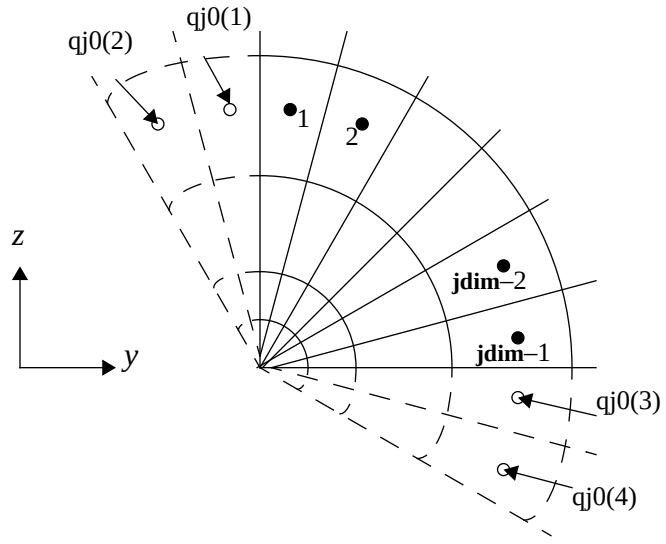

Figure 6-9. Periodic boundary condition (**bctype** 2005) example.

For density and pressure, boundary condition type 2005 sets:

```
qj0(1) = q(jdim-1)
qj0(2) = q(jdim-2)
qj0(3) = q(1)
qj0(4) = q(2)
```
(6-28)

The boundary conditions for the velocity components u, v, and w are assigned similarly, except that they are first rotated through the periodic angle, in this case $90°$. For example,

$$\begin{aligned} u_{1,\mathrm{BC}} &= \mathtt{qj0}(k,i,2,1) = \mathtt{q}(j,k,i,2) \quad \text{(unchanged)} \\ v_{1,\mathrm{BC}} &= \mathtt{qj0}(k,i,3,1) = \mathtt{q}(j,k,i,3)\cos\theta_p - \mathtt{q}(j,k,i,4)\sin\theta_p \\ w_{1,\mathrm{BC}} &= \mathtt{qj0}(k,i,4,1) = \mathtt{q}(j,k,i,3)\sin\theta_p + \mathtt{q}(j,k,i,4)\cos\theta_p \end{aligned} \quad \textbf{(6-29)}$$

For an example of how this boundary condition can be used for linear displacement, see the 2-d vibrating plate sample test case in Section 9.1.6 on page 177.

6.2.5 Radial Pressure Equilibrium

bctype 2006

Boundary condition type 2006 is a cell-center type boundary condition. Radial pressure equilibrium is used as a downstream boundary condition when it is desired to specify a pressure, but a large swirling component is present in the flow. Typically, this boundary condition would be used in turbomachinery applications in which a swirling flow is established by a rotor that is not corrected by the presence of stators or exit guide vanes.

Four additional pieces of information are needed for this boundary condition type (**ndata** = 4). The grid number (**ngridc**) of the grid *from* which the integration of pressure is to be continued is specified. If the integration in this grid is not continued from another grid, input 0. A value for $\tilde{p}/\tilde{p}_\infty$ is input. If **ngridc** = 0, this value will be the starting value for the integration. If **ngridc** 0, then the pressure value from the connecting point in grid **ngridc** is used as the starting value for the integration in the current block. The direction in which integration is to proceed is specified with the parameter **intdir**. It may have an absolute value of 1, 2, or 3 for integration in the i, j, or k directions, respectively. The sign of **intdir** indicates whether the integration proceeds in the increasing (positive) or decreasing (negative) coordinate direction. The **intdir** direction *must* be the radial direction. The fourth piece of information needed is the physical direction along which the radial axis lies, denoted with the parameter **axcoord**. It may have the values of +1, +2, or +3 for a radial axis aligned with the x, y, or z axes. The input lines for this boundary condition type would look like:

```
ngridc        P/Pinf   intdir  axcoord
```

```
0          0.9        +3         1
```

The radial equilibrium condition requires that the pressure satisfy

$$\frac{dp}{dr} = \rho\frac{(V_\theta)^2}{r} \tag{6-30}$$

where V_θ is the circumferential velocity component and r is the radius. The pressure ratio, $\tilde{p}/\tilde{p}_\infty$, is set at either the bottom or the top of the block face and then the radial equilibrium equation is integrated in either the increasing or decreasing radial direction to give the pressure at all other radii. The trapezoidal rule is used to perform the integration. The other flow-field variables (ρ, u, v, w) are extrapolated from inside the computational domain. See "Extrapolation" on page 84.

This boundary condition assumes that one coordinate direction on the block face is essentially radial and the other is essentially circumferential and that the integration is being carried out in the radial direction. Since there is no way to verify this in the code, care *must* be exercised when using this boundary condition to insure that this and the following restrictions are met. Continuation of the radial equilibrium condition through block boundaries is restricted to blocks that have the *same* orientation. For example, if the equilibrium condition is to be continued through a k boundary from an adjacent block, then i and j must run in the same direction in both blocks. This also implies that the k boundary must be $k = 1$ in one block and $k =$ **kdim** in the second (i.e. the boundary can not be $k = 1$ in both blocks). Also, the segment must run the entire length of the block face in the direction in which the integration is being carried out. For example, if the integration is being carried out in the k direction, then **ksta** must be set to 1 and **kend** must be set to **kdim**. This restriction applies only if the integration is being continued from another block.

Consider a case in which boundary condition 2006 is to be applied at an $i =$ **idim** face, with j = constant radial lines (i.e. lines of constant angle θ) and k = constant circumferential lines (i.e. lines of constant radius). Assume the block face lies in an x = constant plane, as shown in Figure 6-10.

Assuming integration in the $+k$ direction (**intdir** = 3), the pressure (`l` = 5) in the first plane of ghost cells is obtained from

$$\texttt{qi0(j,1,5,3)} = \left(\frac{\tilde{p}}{\tilde{p}_\infty}\right)p_\infty$$

$$\texttt{qi0(j,k,5,3)} = p_{k,j} = p_{k-1,j} + \frac{\rho_{k,j}[(V_\theta)^2]_{k,j}}{\bar{r}_{k,j}}\Delta r_{k,j} \qquad k = 2, \mathbf{kdim} - 1 \tag{6-31}$$

where

$$[(V_\theta)^2]_{k,j} = \mathbf{V}_{k,j} \cdot \overline{\nabla\eta_j}$$
$$\overline{\nabla\eta_j} = \frac{1}{2}[\nabla\eta_{k,j+1/2} + \nabla\eta_{k,j-1/2}]_{\mathbf{idim}-1} \quad \text{(6-32)}$$

$\mathbf{V}_{k,j}$ is the velocity at the cell center $(j, k, \mathbf{idim} - 1)$ and $\bar{r}_{k,j}$ is the average radius in cell (k, j):

$$[r_{k-1/2,j} + r_{k+1/2,j}]/2 \quad \text{(6-33)}$$

(k, j) denotes a cell-center value. $(k \pm 1/2,\ j \pm 1/2)$ denotes a face-center value.

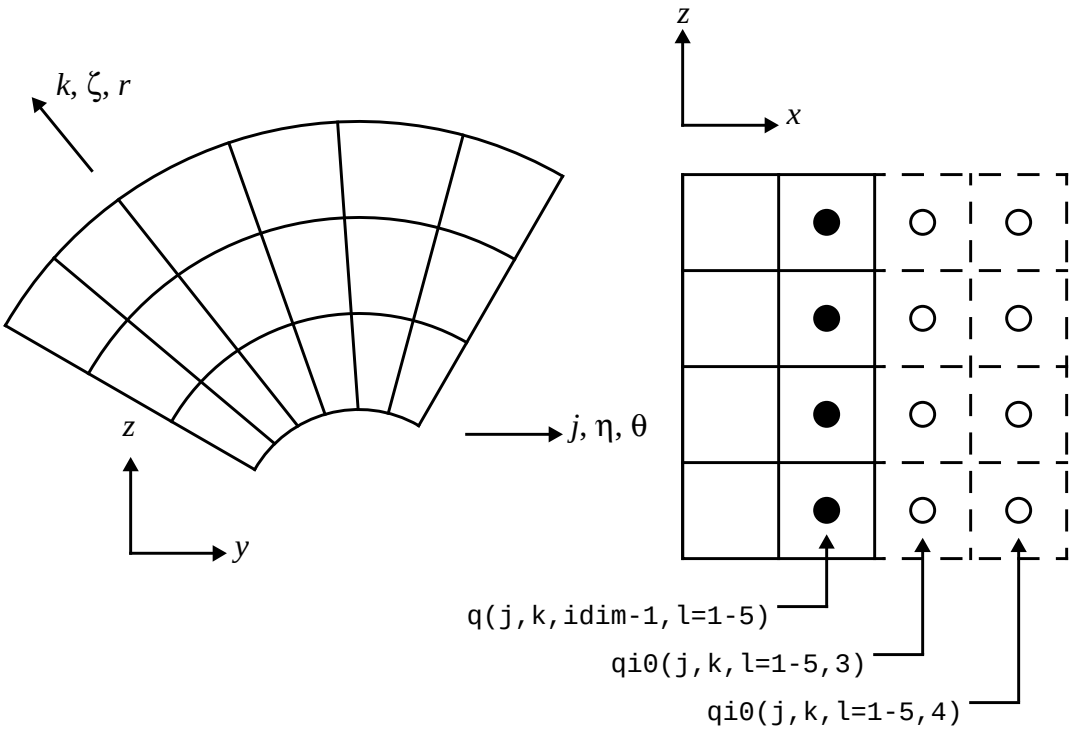

Figure 6-10. Radial pressure equilibrium boundary condition (**bctype** 2006) example.

The velocity and density in the first layer of ghost cells is determined by zeroth order extrapolation from the adjacent interior points. For the case shown in Figure 6-10, the density (`l` = 1) is obtained from

```
qi0(j,k,1,3) = q(j,k,idim-1,1)
```
(6-34)

All flow quantities in the second layer of ghost cells are obtained by zeroth order extrapolation from the first layer of ghost cells:

```
qi0(j,k,l=1-5,4) = qi0(j,k,l=1-5,3)
```
(6-35)

6.2.6 Specify All Primitive Variables

bctype 2007

Boundary condition type 2007 is a cell-center type boundary condition. It involves setting the boundary conditions with the five (**ndata** = 5) primitive variables, using standard CFL3D normalizations: $\tilde{\rho}/\tilde{\rho}_\infty$, $\tilde{u}/\tilde{a}_\infty$, $\tilde{v}/\tilde{a}_\infty$, $\tilde{w}/\tilde{a}_\infty$, and $\tilde{p}/[\tilde{\rho}_\infty(\tilde{a}_\infty)^2]$. An example of the input lines is:

```
rho/rho_inf      u/a_inf      v/a_inf      w/a_inf     p/(rho_inf*a_inf**2)
        1.0          0.3          0.0          0.0                     0.71
```

Note that the input pressure is *not* $\tilde{p}/\tilde{p}_\infty$ but rather $\tilde{p}/(\gamma\tilde{p}_\infty)$. Also note that the turbulence quantities are *not* currently allowed to be specified in the same way as the **q**'s. Instead, they are extrapolated from the interior when **bctype** 2007 is employed.

6.2.7 Specified Pressure Ratio As Sinusoidal Time Function

bctype 2102

The specified pressure ratio as a sinusoidal function of time boundary conditions are cell-center type boundary conditions. The pressure ratio, $\tilde{p}/\tilde{p}_\infty$, is specified as a sinusoidal function of time. The other flow-field variables (ρ, u, v, w) are extrapolated from inside the computational domain. Three pieces of additional information are specified on input (**ndata** = 4): the desired baseline (steady) pressure ratio ($\tilde{p}/\tilde{p}_\infty$), the amplitude of the pressure oscillation ($\Delta\tilde{p}/\tilde{p}_\infty$), the reduced frequency of the pressure oscillation (k_r), and the "grid equivalent" of the dimensional reference length used to define the reduced frequency (L_{ref}). The pressure will vary as

$$\gamma p = \frac{\tilde{p}}{\tilde{p}_\infty} + \frac{\Delta\tilde{p}}{\tilde{p}_\infty}\sin(2\pi k_r t) \qquad \textbf{(6-36)}$$

The reduced frequency is non-dimensionalized by

$$k_r = \frac{\tilde{f}\tilde{L}}{\tilde{a}_\infty} \qquad \textbf{(6-37)}$$

where $\tilde{f}$ is the frequency in cycles per second, $\tilde{L}$ is a characteristic length and $\tilde{a}_\infty$ is the free-stream speed of sound. (Note that this definition of reduced frequency differs from

the "standard" definition $k_r = \tilde{f}\tilde{L}/|\tilde{\mathbf{V}}|_\infty$.) An example of the lines in the input file for this boundary condition is:

```
p/pinf   deltap/pinf       rfreqp          lref
 0.910         0.001       175.93           1.0
```

6.2.8 Variable Data for the 2000 Series

To use this option, set **ndata** = –(the **ndata** used for the constant data value(s) as described above). Then, instead of the line containing the constant data value(s), substitute a line with the name (up to 60 characters) of a formatted file that has the appropriate array of data values. The data file will then be read with the following format:

```
read(iunit,*)                  header/title
read(iunit,*) mdim,ndim,np     mdim,ndim = cell-center dimensions of segment
                               np = number of planes of ghost cell data
read(iunit,*) nvalues          number of data values; nvalues = abs(ndata)
read(iunit,*)((((bcdata(m,n,ip,l),m=1,mdim),n=1,ndim),ip=1,np),
                 l=1,nvalues)
```

The roles of `m,n` vary depending on which face the segment is located:

`m, n` $\rightarrow j, k$	on the i = 1 and the i = **idim** faces where `mdim` = **jend**–**jsta** and `ndim` = **kend**–**ksta**
`m, n` $\rightarrow k, i$	on the j = 1 and the j = **jdim** faces where `mdim` = **kend**–**ksta** and `ndim` = **iend**–**ista**
`m, n` $\rightarrow j, i$	on the k = 1 and the k = **kdim** faces where `mdim` = **jend**–**jsta** and `ndim` = **iend**–**ista**

Note that zeroes are *not* acceptable for `mdim` or `ndim`. Use the actual values of **jdim**-1, **kdim**-1, and/or **idim**-1 for the full face. Only boundary condition type 2007 can make use of two planes of data. For all other boundary conditions, set `np` = 1.

6.3 Block Interface Boundary Conditions

For all the types of block interface boundary conditions, set **bctype** = 0. If **bctype** 0, but block interface boundary conditions are set, the block interface boundary conditions will supersede. All block interface boundary conditions (one-to-one, patched, and overset) are cell-center type boundary conditions.

6.3.1 One-to-One Blocking

(Input Line Types Twenty-Four through Twenty-Six)

One-to-one (1-1) blocking, sometimes called C^0 continuous, means that the faces shared by two grids are exactly (to machine zero) the same. Several examples showing a variety of blocking strategies are discussed below. In the examples, the variable, **l**, represents the five flow-field variables. The sample inputs illustrate how the index ranges (**ista** to **iend**, **jsta** to **jend**, **ksta** to **kend**) in "LT25 - 1-1 Blocking Connections" on page 37 are assigned so that the correct communication between blocks is established. A good initial check to determine if the one-to-one blocking input is set up correctly is to compare the quantity of points in the range. For example, if two grids share a common portion of a j = constant face, the following must be true:

$$[\mathbf{jend} - \mathbf{jsta} + 1]_{\text{grid 1}} = [\mathbf{jend} - \mathbf{jsta} + 1]_{\text{grid 2}} \quad \textbf{(6-38)}$$

Keep in mind that j = constant faces can communicate with i = constant and/or k = constant faces as well (and vice versa), in which case the check will be

$$[\mathbf{jend} - \mathbf{jsta} + 1]_{\text{grid 1}} = [\mathbf{iend} - \mathbf{ista} + 1]_{\text{grid 2}} \quad \textbf{(6-39)}$$

or

$$[\mathbf{jend} - \mathbf{jsta} + 1]_{\text{grid 1}} = [\mathbf{kend} - \mathbf{ksta} + 1]_{\text{grid 2}} \quad \textbf{(6-40)}$$

respectively. Note that Equation (6-38) through Equation (6-40) are necessary for a one-to-one interface to be specified correctly, but they are not sufficient; the direction in which the indices are input must be correct as well. (See Example 3 on page 106.)

Example 1

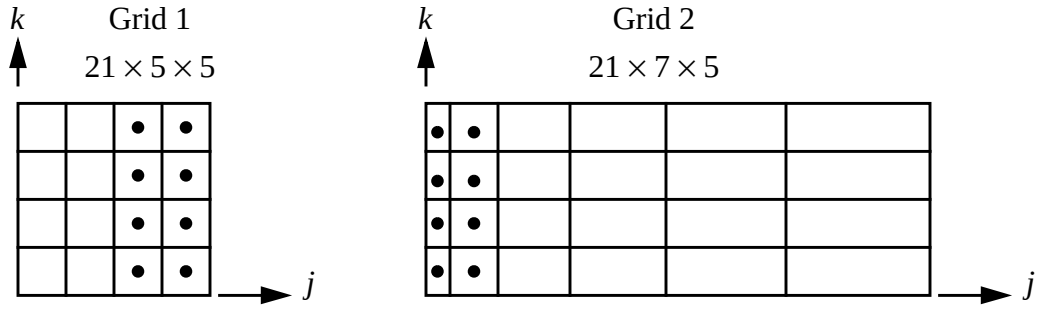

Figure 6-11. One-to-one blocking example 1.

As a simple illustration of 1-1 blocking, consider Figure 6-11. The figure shows an i = constant (not necessarily the *same* constant) face of two grids. Suppose communica-

tion is desired between the j = **jdim** face of grid 1 and the j = 1 face of grid 2. The pertinent input would look something like:

Line Type

```
 6       NGRID  NPLOT3D   NPRINT   NWREST     ICHK      I2D   NTSTEP      ITA
             2        1        0      100        0        0        1        1

 8        IDIM     JDIM     KDIM
            21        5        5
            21        7        5

13        GRID    NBCI0  NBCIDIM    NBCJ0  NBCJDIM    NBCK0  NBCKDIM    IOVRLP
             1        1        1        1        1        1        1         0
             2        1        1        1        1        1        1         0

16    J0: GRID  SEGMENT   BCTYPE     ISTA     IEND     KSTA     KEND     NDATA
             1        1     1003        1       21        1        5         0
             2        1        0        1       21        1        5         0
17  JDIM: GRID  SEGMENT   BCTYPE     ISTA     IEND     KSTA     KEND     NDATA
             1        1        0        1       21        1        5         0
             2        1     1003        1       21        1        5         0

         1-1 BLOCKING DATA:
24        NBLI
             1
25      NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
             1    1          1      5      1     21      5      5      1      3
26      NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
             1    2          1      1      1     21      1      5      1      3
```

Note that in the sample input, both grids have the same value for **idim**. This does not *have* to be true. One grid can share only a portion of a face with another grid.

The boundary conditions at the j = **jdim** face on any i plane, denoted i1, of grid 1 would be set as:

Grid 1	Grid 2
qj0(1,i1,**l**,3) =	q(1,1,i2,**l**)
qj0(2,i1,**l**,3) =	q(1,2,i2,**l**)
qj0(3,i1,**l**,3) =	q(1,3,i2,**l**)
qj0(4,i1,**l**,3) =	q(1,4,i2,**l**)
qj0(1,i1,**l**,4) =	q(2,1,i2,**l**)
qj0(2,i1,**l**,4) =	q(2,2,i2,**l**)
qj0(3,i1,**l**,4) =	q(2,3,i2,**l**)
qj0(4,i1,**l**,4) =	q(2,4,i2,**l**)

The boundary conditions at the $j = 1$ face on any i plane, denoted i2, of grid 2 would be set as:

Grid 2	Grid 1
qj0(1,i2,**l**,1) =	q(4,1,i1,**l**)
qj0(2,i2,**l**,1) =	q(4,2,i1,**l**)
qj0(3,i2,**l**,1) =	q(4,3,i1,**l**)
qj0(4,i2,**l**,1) =	q(4,4,i1,**l**)
qj0(1,i2,**l**,2) =	q(3,1,i1,**l**)
qj0(2,i2,**l**,2) =	q(3,2,i1,**l**)
qj0(3,i2,**l**,2)=	q(3,3,i1,**l**)
qj0(4,i2,**l**,2) =	q(3,4,i1,**l**)

Example 2

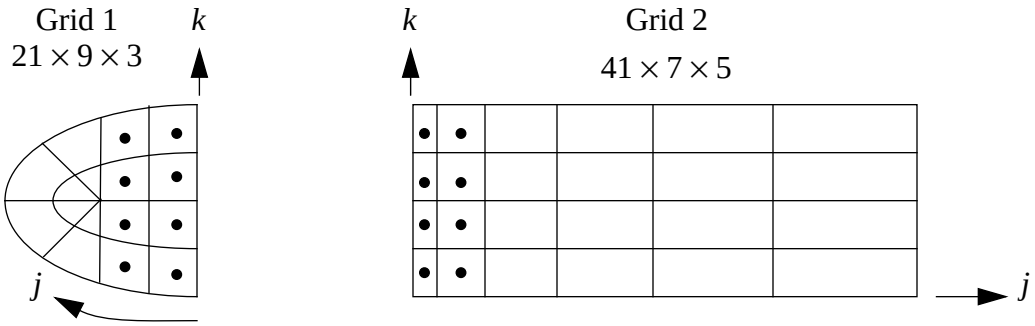

Figure 6-12. One-to-one blocking example 2.

A slightly more complicated example of 1-1 blocking is shown in Figure 6-12. Again, the figure shows an i = constant face (not necessarily the *same* constant) of two grids. In grid 1, j is now a circumferential direction. Communication is desired between the $j = 1$ *and* the $j =$ **jdim** faces of grid 1 and the $j = 1$ face of grid 2. (In this example, it is assumed that the $k = 1$ boundary of grid 1 is a solid wall flat plate, so there is no 1-1 connectivity there.) The pertinent input would look something like:

Line Type

```
 6       NGRID   NPLOT3D   NPRINT   NWREST     ICHK      I2D   NTSTEP      ITA
             2         1        0      100        0        0        1        1

 8        IDIM      JDIM     KDIM
            21         9        3
            41         7        5

13        GRID     NBCI0  NBCIDIM    NBCJ0  NBCJDIM    NBCK0  NBCKDIM    IOVRLP
             1         1        1        1        1        1        1         0
             2         1        1        2        1        1        1         0

16     J0: GRID   SEGMENT   BCTYPE     ISTA     IEND     KSTA     KEND     NDATA
             1         1        0        1       21        1        3         0
             2         1        0        1       21        1        5         0
             2         2     1001       21       41        1        5         0
17  JDIM: GRID   SEGMENT   BCTYPE     ISTA     IEND     KSTA     KEND     NDATA
```

```
          1        1        0        1       21        1        3        0
          2        1     1003        1       41        1        5        0

      1-1 BLOCKING DATA:
24     NBLI
          2
25  NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
          1    1          1      1      1     21      1      3      1      3
          2    1          1      9      1     21      9      3      1      3
26  NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
          1    2          1      1      3     21      1      1      1      3
          2    2          1      1      3     21      1      5      1      3
```

Note how k ranges on grid 2 to coincide with the appropriate k points on the j = 1 and j = **jdim** faces of grid 1. This sample input also shows how grid 1 can share only a portion of the j = 1 face of grid 2 in the i direction.

The boundary conditions at the j = 1 face on any i plane, denoted i1, of grid 1 would be set as:

Grid 1	Grid 2
qj0(1,i1,**l**,1) =	q(1,2,i2,**l**)
qj0(2,i1,**l**,1) =	q(1,1,i2,**l**)
qj0(1,i1,**l**,2) =	q(2,2,i2,**l**)
qj0(2,i1,**l**,2) =	q(2,1,i2,**l**)

The boundary conditions at the j = **jdim** face for the i1 plane of grid 1 would be set as:

Grid 1	Grid 2
qj0(1,i1,**l**,3) =	q(1,3,i2,**l**)
qj0(2,i1,**l**,3) =	q(1,4,i2,**l**)
qj0(1,i1,**l**,4) =	q(2,3,i2,**l**)
qj0(2,i1,**l**,4) =	q(2,4,i2,**l**)

The boundary conditions at the j = 1 face on any i plane, denoted i2, of grid 2 would be set as:

Grid 2	Grid 1
qj0(1,i2,**l**,1) =	q(1,2,i1,**l**)
qj0(2,i2,**l**,1) =	q(1,1,i1,**l**)
qj0(3,i2,**l**,1) =	q(8,1,i1,**l**)
qj0(4,i2,**l**,1) =	q(8,2,i1,**l**)
qj0(1,i2,**l**,2) =	q(2,2,i1,**l**)
qj0(2,i2,**l**,2) =	q(2,1,i1,**l**)
qj0(3,i2,**l**,2) =	q(7,1,i1,**l**)
qj0(4,i2,**l**,2) =	q(7,2,i1,**l**)

Example 3

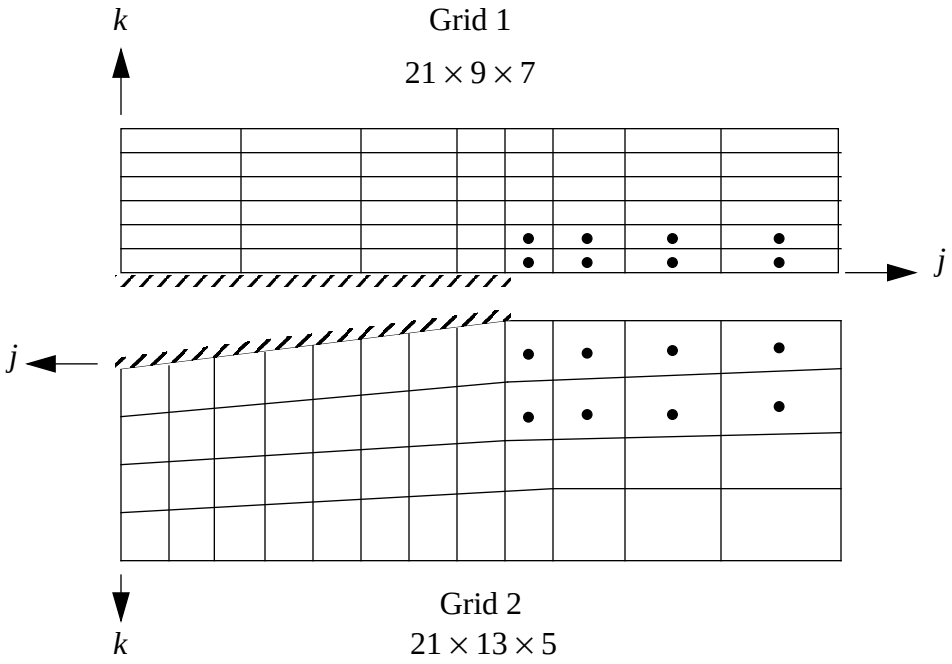

Figure 6-13. One-to-one blocking example 3.

The previous two examples show 1-1 blocking over the entire k range of two grids. In some cases, only segments of faces will utilize 1-1 communication between grids. In Figure 6-13, two grids share a portion of a face beyond boundaries defined as inviscid surfaces. Again, the figure shows an i = constant face (not necessarily the *same* constant) of two grids. Communication is desired between **jsta** = 5 and **jend** = 9 on the k = 1 face of grid 1 and **jsta** = 1 and **jend** = 5 on the k = 1 face of grid 2. The pertinent input would look something like:

Line Type

```
 6       NGRID  NPLOT3D   NPRINT   NWREST     ICHK      I2D   NTSTEP      ITA
             2        1        0      100        0        0        1        1

 8        IDIM     JDIM     KDIM
            21        9        7
            21       13        5

13        GRID     NBCI0  NBCIDIM     NBCJ0  NBCJDIM     NBCK0  NBCKDIM    IOVRLP
             1        1        1        1        1        2        1        0
             2        1        1        1        1        2        1        0

18    K0: GRID  SEGMENT   BCTYPE     ISTA     IEND     JSTA     JEND     NDATA
             1        1     1005        1       21        1        5        0
             1        2        0        1       21        5        9        0
             2        1        0        1       21        1        5        0
             2        2     1005        1       21        5       13        0

      1-1 BLOCKING DATA:
```

```
24        NBLI
            1
25      NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
            1    1           1      5      1     21      9      1      1      2
26      NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
            1    2           1      5      1     21      1      1      1      2
```

Note that, since the j index runs in opposite directions on the two grids, one range is increasing and one range is decreasing in the 1-1 blocking input. It does not matter which grid's range increases and which one decreases as long as they map the points in the correct order.

The boundary conditions at the k = 1 face from **jsta** to **jend**-1 on any i plane, denoted i1, of grid 1 would be set as:

Grid 1	Grid 2
qk0(5,i1,**l**,1) =	q(4,1,i2,**l**)
qk0(6,i1,**l**,1) =	q(3,1,i2,**l**)
qk0(7,i1,**l**,1) =	q(2,1,i2,**l**)
qk0(8,i1,**l**,1) =	q(1,1,i2,**l**)
qk0(5,i1,**l**,2) =	q(4,2,i2,**l**)
qk0(6,i1,**l**,2) =	q(3,2,i2,**l**)
qk0(7,i1,**l**,2) =	q(2,2,i2,**l**)
qk0(8,i1,**l**,2) =	q(1,2,i2,**l**)

The boundary conditions at the k = 1 face from **jsta** to **jend**-1 on any i plane, denoted i2, of grid 2 would be set as:

Grid 2	Grid 1
qk0(1,i2,**l**,1) =	q(8,1,i1,**l**)
qk0(2,i2,**l**,1) =	q(7,1,i1,**l**)
qk0(3,i2,**l**,1) =	q(6,1,i1,**l**)
qk0(4,i2,**l**,1)=	q(5,1,i1,**l**)
qk0(1,i2,**l**,2) =	q(8,2,i1,**l**)
qk0(2,i2,**l**,2)=	q(7,2,i1,**l**)
qk0(3,i2,**l**,2) =	q(6,2,i1,**l**)
qk0(3,i2,**l**,2) =	q(5,2,i1,**l**)

Example 4

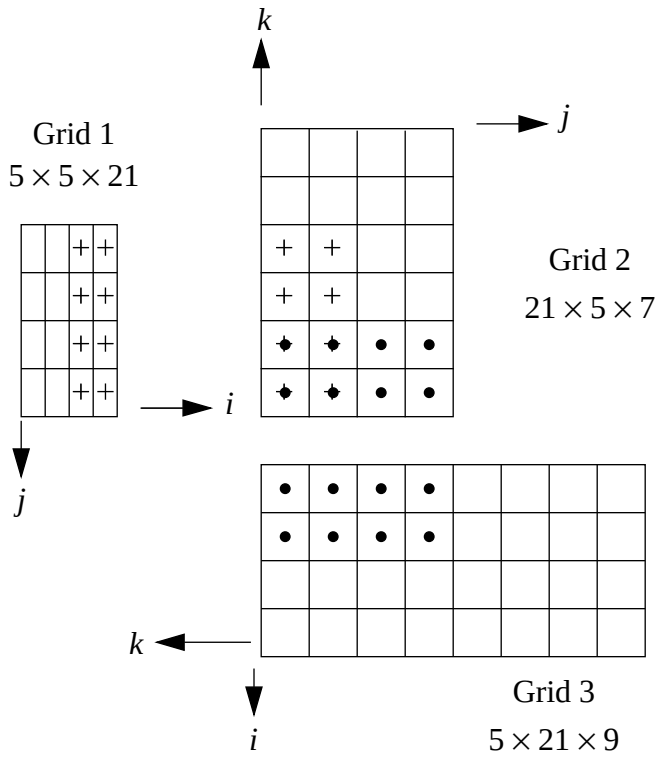

Figure 6-14. One-to-one blocking example 4.

The last example of 1-1 blocking involves three grids shown in Figure 6-14. In the figure, a k = constant (denoted k1) plane of grid 1, an i = constant (denoted i2) plane of grid 2, and a j = constant (denoted j3) plane of grid 3 are shown. Communication between grids 1 and 2 is desired between the i = **idim** face of grid 1 and the j = 1 face of grid 2. Communication between grids 2 and 3 is desired between the k = 1 face of grid 2 and the i = 1 face of grid 3. The pertinent input would look something like:

Line Type

```
 6      NGRID  NPLOT3D   NPRINT   NWREST     ICHK      I2D   NTSTEP      ITA
            3        1        0      100        0        0        1        1

 8       IDIM     JDIM     KDIM
            5        5       21
           21        5        7
            5       21        9

13       GRID     NBCI0  NBCIDIM    NBCJ0  NBCJDIM    NBCK0  NBCKDIM   IOVRLP
            1        1        1        1        1        1        1        0
            2        1        1        2        1        1        1        0
            3        2        1        1        1        1        1        0
14   I0: GRID  SEGMENT    BCTYPE      JSTA     JEND     KSTA     KEND    NDATA
            1        1     1003         1        5        1       21        0
            2        1     1003         1        5        1        7        0
            3        1     1005         1       21        1        5        0
```

```
            3        2        0        1       21        5        9        0
15IDIM: GRIDSEGMENTBCTYPE   JSTA     JEND     KSTA     KEND    NDATA
       1    1        0        1        5        1       21        0
       2    1     1001        1        5        1        7        0
       3    1     1000        1       21        1        9        0
16   J0: GRID  SEGMENT   BCTYPE     ISTA     IEND     KSTA     KEND     NDATA
            1        1     1000        1        5        1       21         0
            2        1        0        1       21        1        5         0
            2        2     1000        1       21        5        7         0
            3        1     1003        1        5        1        9         0

18   K0: GRID  SEGMENT   BCTYPE     ISTA     IEND     JSTA     JEND     NDATA
            1        1     1003        1        5        1        5         0
            2        1        0        1       21        1        5         0
            3        1     1003        1        5        1       21         0

       1-1 BLOCKING DATA:
24      NBLI
           2
25     NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
           1    1          5      1      1      5      5     21      2      3
           2    2          1      1      1     21      5      1      1      2
26     NUMBER GRID    :   ISTA   JSTA   KSTA   IEND   JEND   KEND  ISVA1  ISVA2
           1    2         21      1      5      1      1      1      3      1
           2    3          1      1      9      1     21      5      2      3
```

Notice that the i index range for grid 2 runs in the opposite direction of the k index range for grid 3 due to the right-hand rule.

The boundary conditions at the i = **idim** face of grid 1 would be set as:

Grid 1	Grid 2
qi0(1,k1,**l**,3) =	q(1,4,i2,**l**)
qi0(2,k1,**l**,3) =	q(1,3,i2,**l**)
qi0(3,k1,**l**,3) =	q(1,2,i2,**l**)
qi0(4,k1,**l**,3) =	q(1,1,i2,**l**)
qi0(1,k1,**l**,4) =	q(2,4,i2,**l**)
qi0(2,k1,**l**,4) =	q(2,3,i2,**l**)
qi0(3,k1,**l**,4) =	q(2,2,i2,**l**)
qi0(4,k1,**l**,4) =	q(2,1,i2,**l**)

The boundary conditions from **ksta** to **kend**-1 on the j = 1 face of grid 2 would be set as:

Grid 2	Grid 1
qj0(1,i2,**l**,1) =	q(4,k1,4,**l**)
qj0(2,i2,**l**,1) =	q(3,k1,4,**l**)
qj0(3,i2,**l**,1) =	q(2,k1,4,**l**)
qj0(4,i2,**l**,1) =	q(1,k1,4,**l**)
qj0(1,i2,**l**,2) =	q(4,k1,3,**l**)
qj0(2,i2,**l**,2) =	q(3,k1,3,**l**)
qj0(3,i2,**l**,2) =	q(2,k1,3,**l**)
qj0(4,i2,**l**,2) =	q(1,k1,3,**l**)

The boundary conditions at the k = 1 face of grid 2 would be set as:

Grid 2	Grid 3
qk0(1,i2,**l**,1) =	q(j3,8,1,**l**)
qk0(2,i2,**l**,1) =	q(j3,7,1,**l**)
qk0(3,i2,**l**,1) =	q(j3,6,1,**l**)
qk0(4,i2,**l**,1) =	q(j3,5,1,**l**)
qk0(1,i2,**l**,2) =	q(j3,8,2,**l**)
qk0(2,i2,**l**,2) =	q(j3,7,2,**l**)
qk0(3,i2,**l**,2) =	q(j3,6,2,**l**)
qk0(4,i2,**l**,2) =	q(j3,5,2,**l**)

The boundary conditions from **ksta** to **kend**-1 on the i = 1 face of grid 3 would be set as:

Grid 3	Grid 2
qi0(j3,5,**l**,1) =	q(4,1,i2,**l**)
qi0(j3,6,**l**,1) =	q(3,1,i2,**l**)
qi0(j3,7,**l**,1) =	q(2,1,i2,**l**)
qi0(j3,8,**l**,1) =	q(1,1,i2,**l**)
qi0(j3,5,**l**,2) =	q(4,2,i2,**l**)
qi0(j3,6,**l**,2) =	q(3,2,i2,**l**)
qi0(j3,7,**l**,2) =	q(2,2,i2,**l**)
qi0(j3,8,**l**,2) =	q(1,2,i2,**l**)

6.3.2 Patched-Grid Interpolation

(Input Line Type Twenty-Seven)

6.3.2.1 General Information

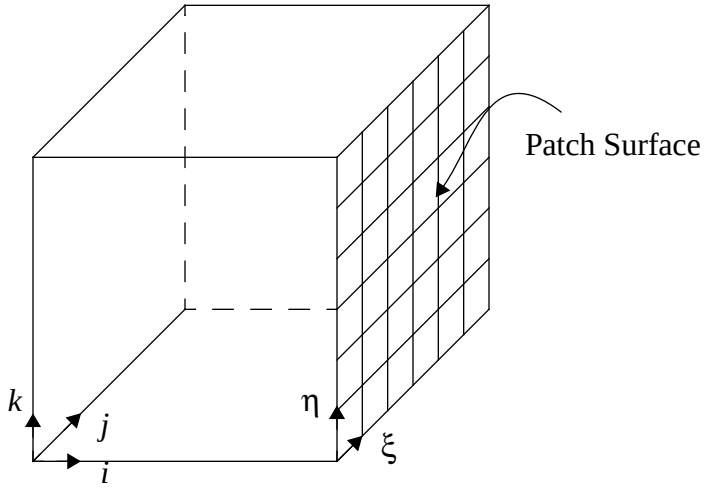

Figure 6-15. Patched-grid surface.

This section describes the "basics" of patching in CFL3D and then shows a simple static patching example to illustrate the basic concepts. Next, an example of dynamic patching is given. Patched-grid interpolation is designed for communication between grids that share a common face, but are *not* C^0 continuous. For example, a global (3-d) grid is represented by the "box" in Figure 6-15. The grid must be right-handed as shown with the i, j, and k axes drawn in the figure. Patching may occur on an i = constant, j = constant, or k = constant surface. In the figure, an i = constant surface is indicated as the patched surface. Note that the adjacent grid involved in the patch is not shown in the figure. Patching works best when the spacing of the adjacent grid in the normal direction to the patch is the same as that in the other grid. While the patch surface is shown as a plane, it may be non-planar as well. The local (2-d) indexing on the patch surface is indicated by the ξ and η axes.

The global and local indices correspond as follows (this is important when determining the input parameters for dynamic patching in Section 3.42 on page 49):

If the patch surface is on an i = constant surface (as shown in Figure 6-15), then

$$\begin{aligned} k &\Leftrightarrow \eta \\ j &\Leftrightarrow \xi \end{aligned} \qquad \text{(6-41)}$$

If the patch surface is on a j = constant surface, then

$$i \Leftrightarrow \eta$$
$$k \Leftrightarrow \xi \qquad \textbf{(6-42)}$$

If the patch surface is on a k = constant surface, then

$$i \Leftrightarrow \eta$$
$$j \Leftrightarrow \xi \qquad \textbf{(6-43)}$$

One grid may have multiple grids patched to it. For example, the flow over a nose cone illustrated in Figure 6-16 shows block 1 patched to both block 2 and block 3. Note that the orientation of grid lines in block 3 is different from that in blocks 1 and 2 to illustrate the features of the patching algorithm.

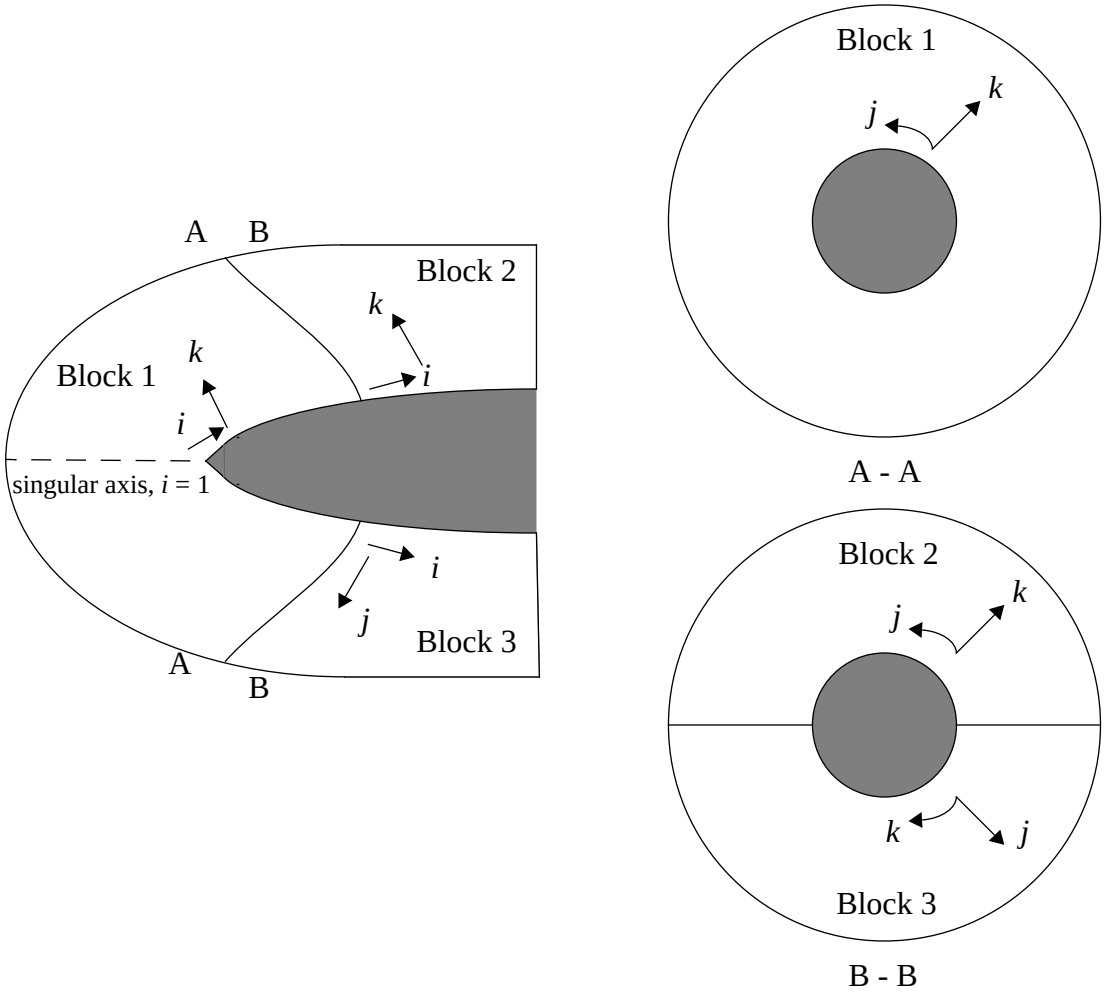

Figure 6-16. Patched-grid example.

To provide two-way communication between all blocks, a total of seven interpolations are required for this case, as follows:

Interpolation:	Description:
1	*To i* = **idim** of block 1 *from i* = 1 of blocks 2 and 3 (two "from" blocks)
2	*To i* = 1 of block 2 *from i* = **idim** of block 1
3	*To i* = 1 of block 3 *from i* = **idim** of block 1
4	*To j* = 1 of block 2 *from k* = 1 of block 3
5	*To j* = **jdim** of block 2 *from k* = **kdim** of block 3
6	*To k* = 1 of block 3 *from j* = 1 of block 2
7	*To k* = **kdim** of block 3 *from j* = **jdim** of block 2

The interpolations may be input in any order (but must be consistent once the order is chosen).

6.3.2.2 Description/Discussion of Input Parameters

Basically, the patch algorithm works as follows. The interpolations are cycled through, one at a time, so that at any given time there is one surface being interpolated to. Note, however, that there may be more than one surface being interpolated from. In order to interpolate from one (or more) block(s) to another, interpolation coefficients are required. These are found by expressing the cell-center coordinates x_c, y_c, z_c on the "to" side of the patch surface in terms of a nonlinear polynomial in ξ, η, where ξ, η are the local coordinates on the "from" side of the patch surface. Newton iteration is used to invert the polynomial to find the local coordinates of the cell center, ξ_c, η_c.

Because of the nonlinearity of the equation, problems may arise in the iteration process. The parameters **ifit**, **limit**, and **itmax** may be adjusted to try to overcome any convergence difficulties. There must be **ninter2** values for each of these three parameters. **Limit** is the maximum step size in ξ or η allowed during the search procedure. The value 1 seems to be a good general choice. **Itmax** is the maximum number of search steps allowed per grid point. A rule of thumb is **limit** $\times$ **itmax** $\approx$ the maximum dimension of i, j, or k on a patch surface.

Ifit controls the order of the polynomial fit used to relate x, y, or z to ξ and η. **Ifit** = 1 for linear in both ξ and η (bilinear). **Ifit** = 2 for quadratic in both ξ and η (degenerate biquadratic). **Ifit** = 3 for quadratic in ξ, linear in η. **Ifit** = 4 for linear in ξ, quadratic in η.

Some tips for choosing **ifit** are:

- A grid with highly curved grid lines in one of the *local* directions will need a quadratic polynomial in that direction.
- Refer to Section 6.3.2.1 and the user's own knowledge of the grid to decide which (if any) of the local directions ξ or η are highly curved.
- Use the lowest order fit which seems reasonable in each direction.

- One-to-one patching can always be done with a bilinear fit, regardless of curvature.

To and **from** are numbers containing attributes of the "to" side of the patch surface and the "from" side of the patch surface. For each of the **ninter2** interpolations there is only one value of **to**, but there may be more than one value of **from** which is set by the parameter **nfb**. (**Nfb** is the number of blocks on the "from" side of the patch surface.) The values of **to** and **from** are of the form:

to/**from** = Nmn

where "N" indicates the block number; "m" indicates the coordinate which is constant on the patch surface (may differ on either side of the patch):

$$m = 1 \Rightarrow i = \text{constant}$$
$$m = 2 \Rightarrow j = \text{constant}$$
$$m = 3 \Rightarrow k = \text{constant}$$

and "n" indicates on which of the two possible m = constant surfaces the patch surface occurs:

$$n = 1 \text{ for patch on } m = 1 \text{ surface}$$
$$n = 2 \text{ for patch on } m = mdim \text{ surface}$$

The following inputs pertain to the nose cone case in Figure 6-16:

Interpolation #	to	nfb	from
1	112	2	211, 311
2	211	1	112
3	311	1	112
4	221	1	331

Example

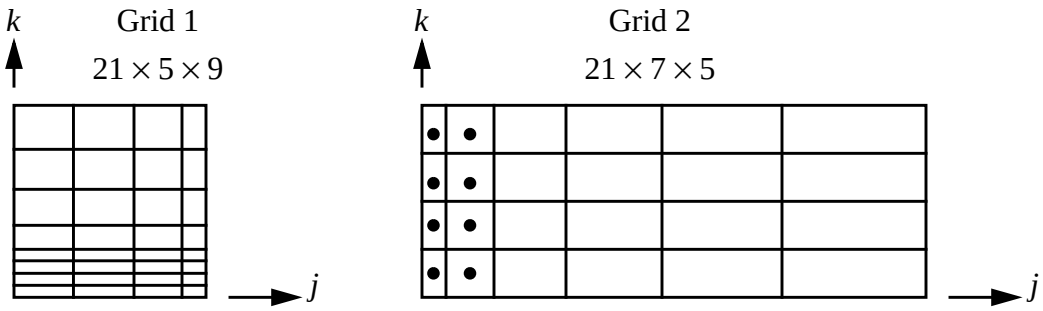

Figure 6-17. Patched-grid example.

Suppose the k dimension of grid 1 in the first example of 1-1 blocking (see Figure 6-11 on page 102) was actually **kdim** = 9. Grid 1 might look something like that drawn in Figure 6-17. As in Figure 6-11, Figure 6-17 shows an i = constant face (not necessarily the *same* constant) of two grids. Notice that the spacing in the direction normal to the interface (i.e. j) is approximately constant across the interface.

In Version 5.0 of CFL3D, patched-grid interpolation data for *static* patched interfaces (interfaces that do not change with time) must be obtained as a preprocessing step. The code ronnie is designed for this task. *Dynamic* patched interfaces, such as occur when grids slide past one another, are computed internally in CFL3D.

Note that, if two grids with very different sizes are patched together, it may be necessary to use a limiter on the gradients there. To do this, replace (hard-wire) the calls to `int2` with calls to `int3` instead.

6.3.3 Chimera Grid Interpolation

(Input Line Type Ten)

Grid overlapping, also known as the overset-grid method or chimera technique, requires neither 1-1 connectivity nor a shared interface to pass flow information from one grid to another. With this method, a variety of grid topologies can be used together. The chimera implementation used in CFL3D is based on the method of Benek et al.[12] As a simplified example of grid overlapping, consider a cross section of a polar grid and a Cartesian mesh as shown Figure 6-18. Suppose the grids overlap as in Figure 6-19. Since both grids cover the same area, computations on both grids in this area would be redundant. Therefore, certain points on the Cartesian grid will be eliminated from the computation. The polar grid is used to carve a "hole" in the Cartesian grid. In this example, the hole is defined as any cell-center point of the Cartesian grid interior to the $k = 4$ grid line of the polar mesh. The Cartesian mesh with the hole carved out is illustrated in Figure 6-20. Any cell center point of the Cartesian grid located within this hole is designated a "hole point". The first two "nonhole" cell-center points of the Cartesian grid that border a hole point both vertically and horizontally are labelled "fringe points". The remaining grid points that have not been designated as either hole or fringe points are called "field" points. Figure 6-21 depicts the hole, fringe, and boundary points for this example. Each fringe point of the Cartesian grid falls within a "target cell" of the polar grid.

A searching algorithm is used to identify the particular eight points that define the hexahedral target cell. The search begins with an initial guess for the target cell. Next, the current target cell is isoparametrically mapped into a unit cube in computational space. The same transformation into the mapped coordinate system is then applied to the fringe point; if the mapped fringe point lies in the same unit cube as the current target cell, then that target cell in fact encases the fringe point. If the mapped fringe point lies outside the unit cube, then the current target cell is not the correct choice. However, the magnitude and direction of the mapped fringe point relative to the current target cell may be used to choose a new guess for the target cell. The mapping process is repeated until the correct target cell is identified. With the correct target cell identified, the data are transferred from the target cell to the fringe point with trilinear interpolation in computational space. Outer boundary values of the polar grid are determined in a similar manner. The MultiGeometry Grid Embedder (MaGGiE) code, written specifically for CFL3D by Baysal et al.[11], is used to determine the interpolation information between grids.

Note that if an overlapped (Chimera) grid is used and there is grid overlap **on** solid surfaces, then forces are double-counted at the overlap. If possible, make appropriate use of **segment** < 0 (see note (2) of Section 3.7) to remedy this.

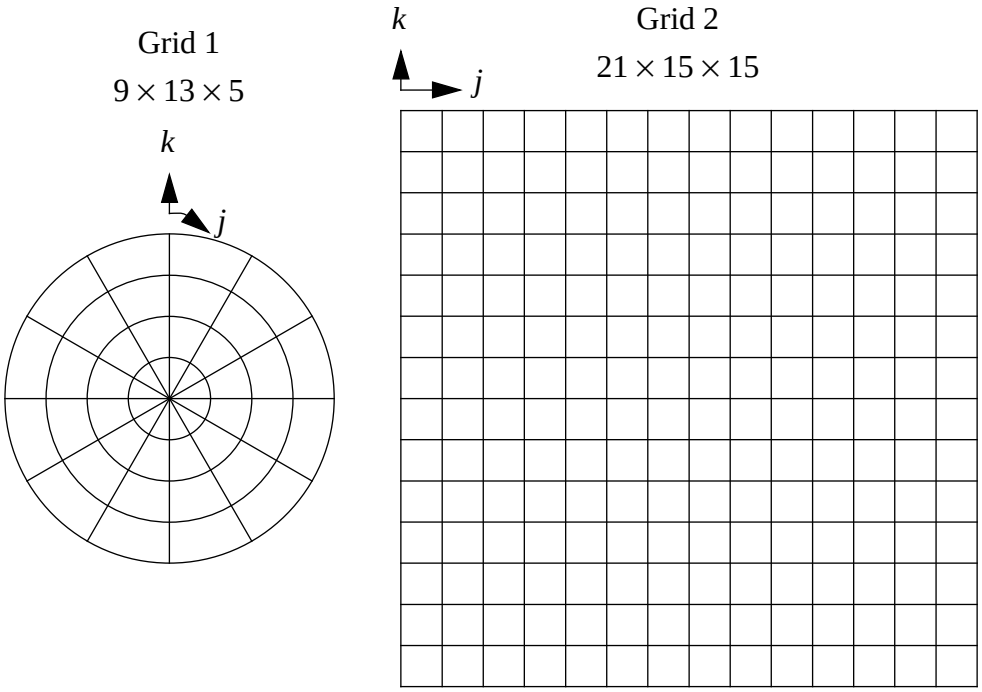

Figure 6-18. Grid-overlapping grid examples.

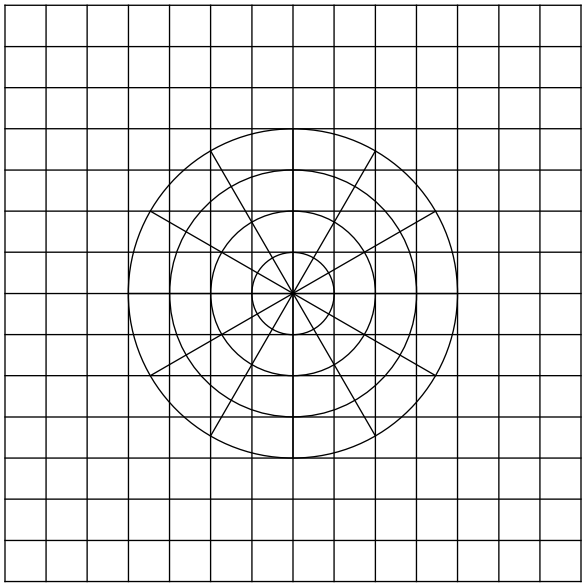

Figure 6-19. Overlapped grids.

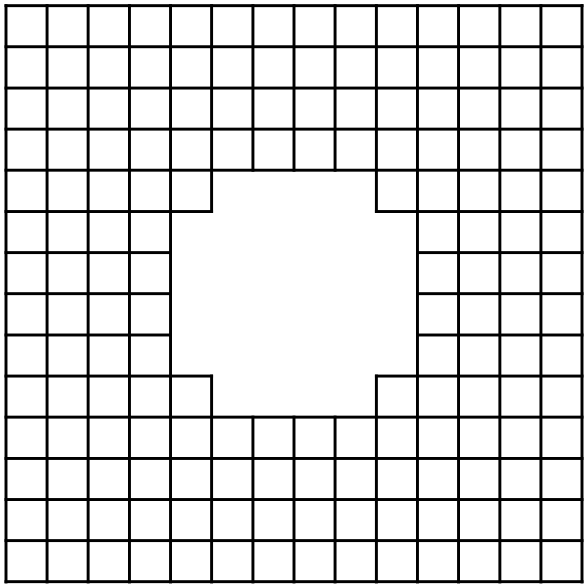

Figure 6-20. Hole in Cartesian grid in region of polar grid.

● Hole Point for Cartesian Grid

\+ Fringe Point for Cartesian Grid

○ Boundary Point for Polar Grid

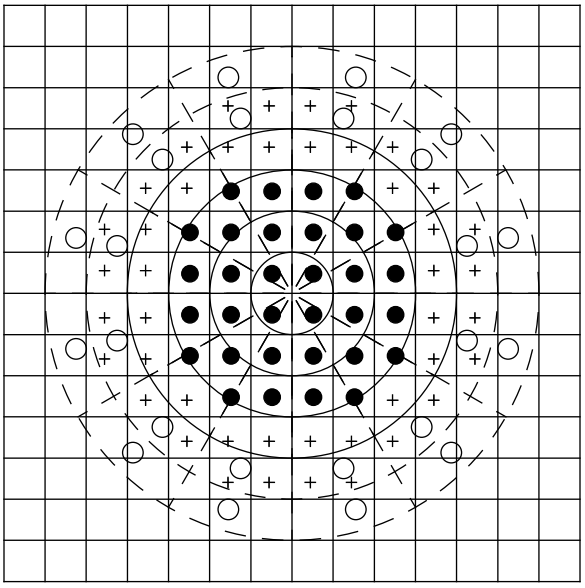

Figure 6-21. Hole, fringe, and boundary points for overlapped grid example.

6.3.4 Embedded Mesh

Regular refinement of coarse mesh

Embedded grids are useful when high gradient areas are limited to an identifiable region (see reference 25). An embedded grid can be placed in that region to resolve the flow field without refining the entire mesh. As an example, an embedded grid scheme is shown in Figure 6-22 for two dimensions. The diagram represents full refinement in both directions. The solid lines define a finer mesh embedded completely within a coarser mesh depicted by the dashed lines. In the figure, a portion of the flow field is covered by both the embedded mesh and a portion of the coarser grid. The grids are coupled together during the solution process. The cell-center variables on a coarser grid cell which underlies a finer embedded grid cell are replaced with a volume-weighted restriction of variables from the four (2-d) or eight (3-d) finer grid cells, similar to the restriction operators used in a global multigrid scheme.

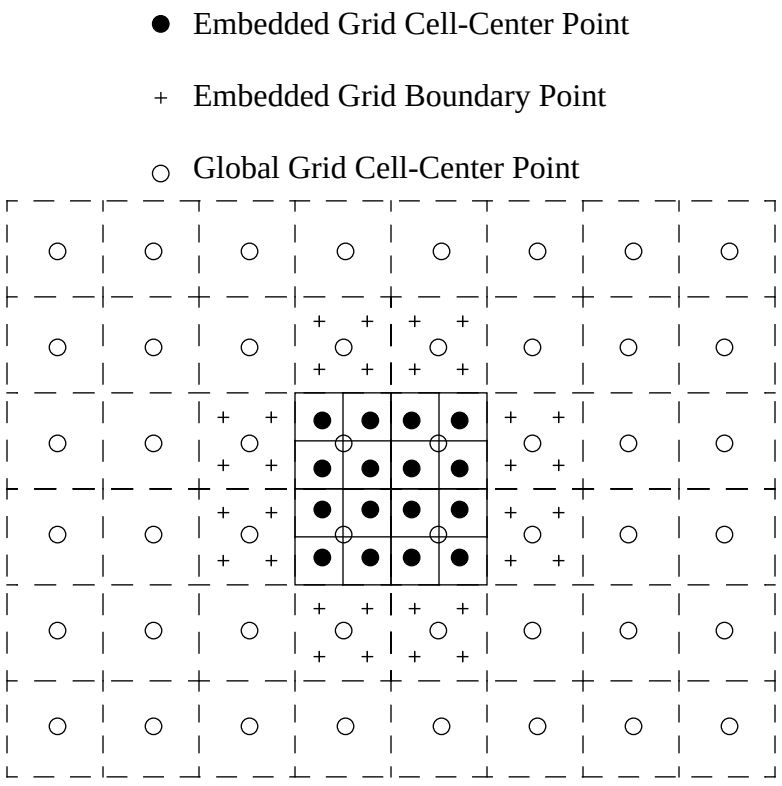

Figure 6-22. Embedded grid example.

For the embedded grid, the computational boundaries occur either at a physical boundary, such as a wing, a symmetry plane, an inflow/outflow plane along a zonal interface (one-to-one, patched, or overset), or along an interior computational surface of a coarser grid. Along an interior surface, two additional lines of data corresponding to an analytical continuation of the finer grid cell centers are constructed from linear interpolation of the coarser grid state variables. An enlarged view of the lower-left corner of the embedded

mesh of Figure 6-22 is shown in Figure 6-23. Arrows indicate which coarse grid points are used in the linear interpolation scheme to obtain a couple of the boundary points drawn. If the input parameter **iconsf** = 1 ("LT20 - Mesh Sequencing and Multigrid"), then global conservation is enforced by replacing the coarse grid flux at an embedded grid boundary with the sum of the finer grid fluxes which share the common interface. If **iconsf** = 0, the coarse grid flux is computed using the volume-weighted restriction from the fine grid, and conservation is not insured.

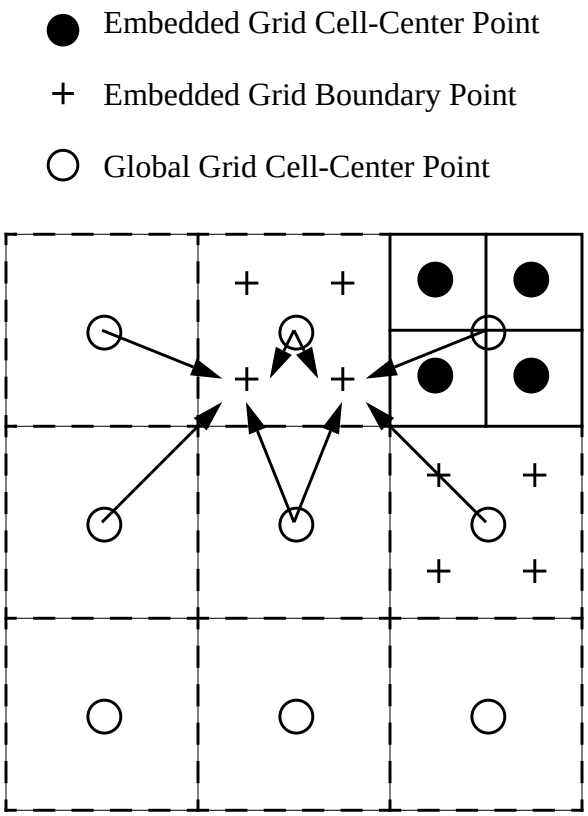

Figure 6-23. Enlarged portion of embedded mesh.

When designing an embedded mesh, it is important to remember that at least two layers of coarse grid cells should surround the embedded boundaries unless the embedded boundary is set with a physical boundary condition (e.g. solid wall, plane of symmetry, or far-field) or a zonal boundary such as a one-to-one, patched, or overset type. In the example of Figure 6-22, there are two layers of coarse grid cells above and below the embedded mesh and three layers of coarse grid cells on the left and right sides of the embedded mesh.

Suppose the grids in Figure 6-22 are in an i = constant plane. Also, **jdim** = 9 and **kdim** = 7 for the coarse grid and **jdim** = 5 and **kdim** = 5 for the embedded mesh. The pertinent input would look something like:

Line Type

```
 6       NGRID   NPLOT3D   NPRINT   NWREST     ICHK      I2D   NTSTEP      ITA
             2         1        0      100        0        0        1        1
 7         NCG       IEM IADVANCE   IFORCE IVISC(I) IVISC(J) IVISC(K)
             0         0        0        0        0        0        0
             0         1        0        0        0        0        0
 8        IDIM      JDIM     KDIM
            21         9        7
            41         5        5

10      INEWG   IGRIDC       IS       JS       KS       IE       JE       KE
             0        0        0        0        0        0        0        0
             0        1        1        4        3       21        6        5

20        MSEQ   MGFLAG   ICONSF      MTT     NGAM
             1        0        1        0        0

22        NCYC   MGLEVG    NEMBL    NITFO
             1        1        1        0
```

In this example, the embedded mesh extends the entire length of the global grid in the i direction. The dimensions of the embedded mesh must satisfy a particular relationship to the data in "LT10 - Embedded Mesh Specifications". The input parameters **is**, **ie**, **js**, **je**, **ks**, and **ke** are indices in the global grid to which the embedded mesh is connected. Thus, the following must be true:

$$\mathbf{idim}_{embedded} = 2(\mathbf{ie} - \mathbf{is})_{global} + 1 \qquad \textbf{(6-44)}$$

Analogous relationships hold for the j and k directions. Note that the only exception to Equation (6-44) occurs in the i direction for 2-d cases, where **is** = 2, **ie** = 1 and $\mathbf{idim}_{embedded} = \mathbf{idim}_{global} = 2$.

There is also an option called "semi-coarsening" in which the number of points in the i direction are the same for the embedded grid and the global grid in the embedded region. This is helpful if there are sufficient points in the i direction, but grid enrichment in $j-k$ planes is desirable. There is an internal check for this option and the only change in the input sample above is to set **idim** = 21 for the embedded grid as well. Note that semi-coarsening can only be used in the i direction.

Another nice option with grid embedding is to add an embedded grid to a previously-run coarse grid problem. For instance, suppose a converged coarse grid solution is obtained and the flow field looks well resolved everywhere except in a vortex region. An embedded grid for that region can be tacked on to the original grid file and the case restarted. On the *first* run of the restart, set **inewg** = 1 for the embedded grid. This lets the code know that a solution not on the current restart file is beginning. On the next run, the embedded grid solution *will* be on the restart file, so set **inewg** = 0.